Model-based Reinforcement Learning

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

avereshc@buffalo.edu

November 19, 2019

Slides are based on Model-Based Policy Learning by Sergey Levine; Model Based Reinforcement Learning by Katerina Fragkiadaki; Model-Based [Deep] Reinforcement Learning by Chelsea Finn

Overview

Table of Contents

Planning

Planning: any computational process that uses a model to create or improve a policy

Model

Planning

Planning: any computational process that uses a model to create or improve a policy

$$\text{Model} \xrightarrow{\text{Planning}}$$

Planning

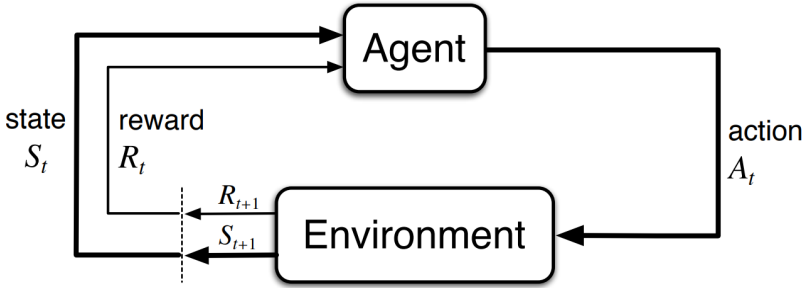
Planning: any computational process that uses a model to create or improve a policy

$$\text{Model} \xrightarrow{\text{Planning}} \text{Policy}$$

Planning

Planning: any computational process that uses a model to create or improve a policy

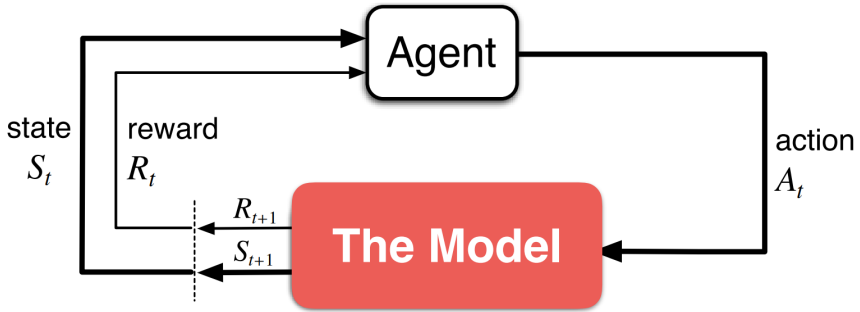
$$\text{Model} \xrightarrow{\text{Planning}} \text{Policy}$$



Planning

Planning: any computational process that uses a model to create or improve a policy

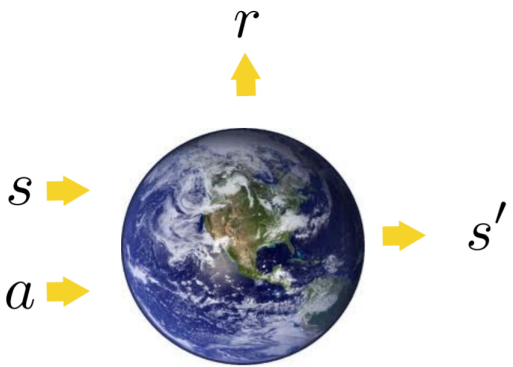
$$\text{Model} \xrightarrow{\text{Planning}} \text{Policy}$$



Model

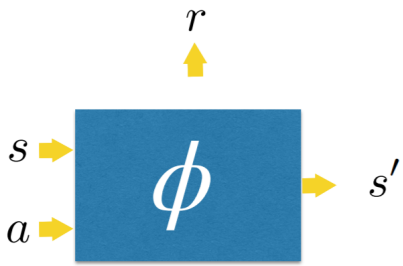
Model

Model is anything the agent can use to predict how the environment will respond to its actions, concretely, the state transition $T(s'|s,a)$ or reward function $r(s,a)$.

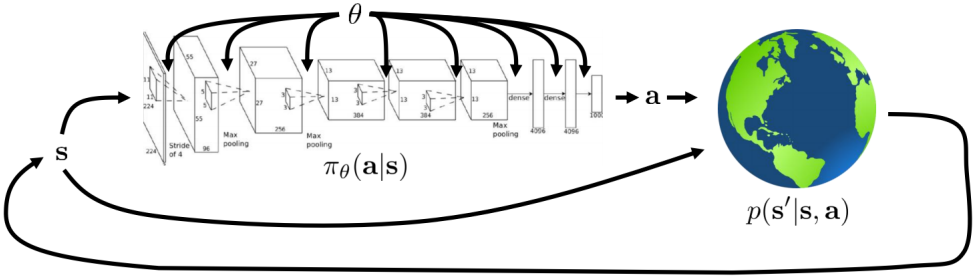


Model Learning

We will be learning the model using experience tuples, that will convert it into a supervised learning problem.

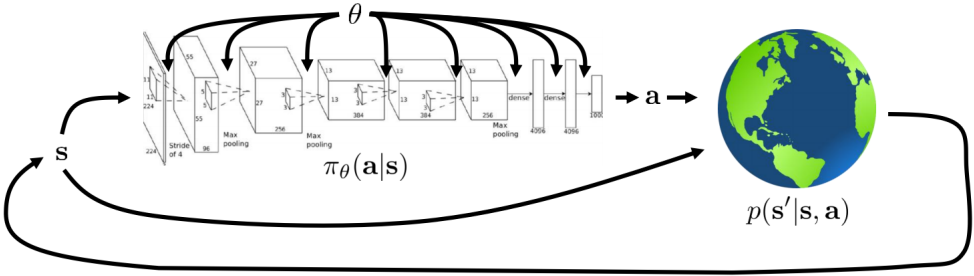


Recap: Reinforcement Learning Objective



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

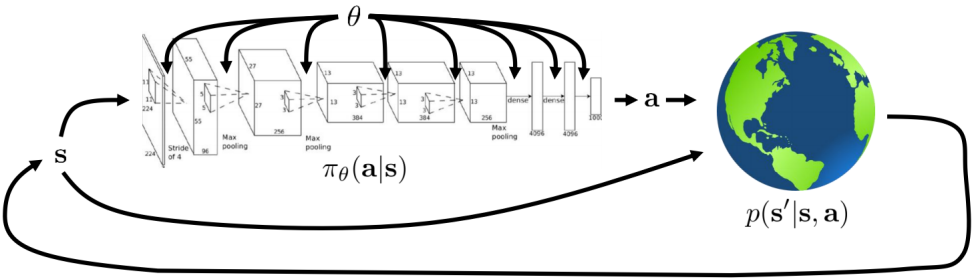
Recap: Reinforcement Learning Objective



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

What is the main objective?

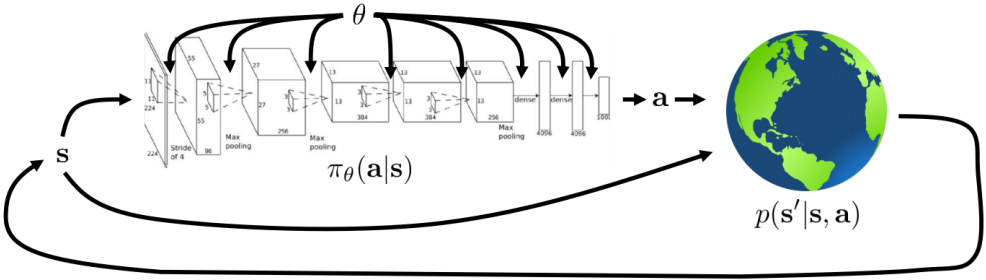
Recap: Reinforcement Learning Objective



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Recap: Model-Free Reinforcement Learning

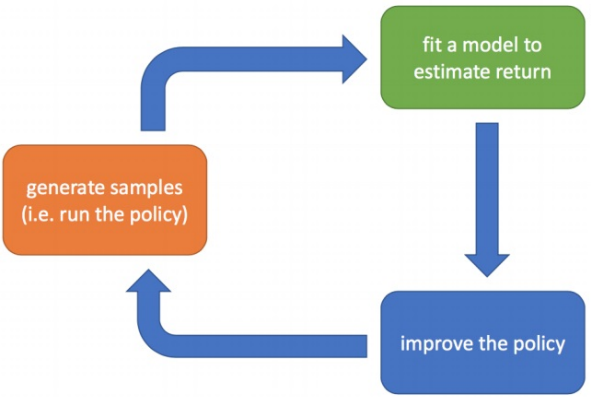


$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}$$

assume this is unknown
don't even attempt to learn it

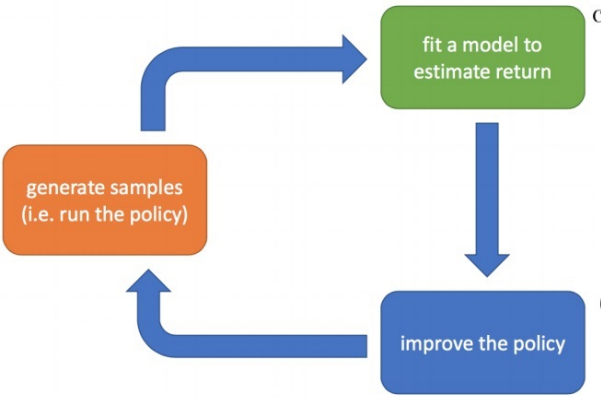
$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Reinforcement Learning Problem



compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

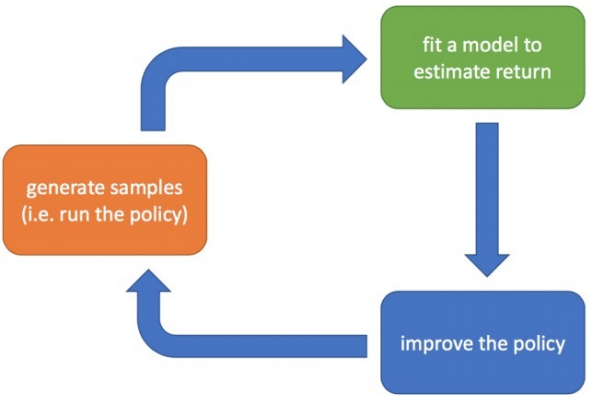
Reinforcement Learning Problem



compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)

Reinforcement Learning Problem

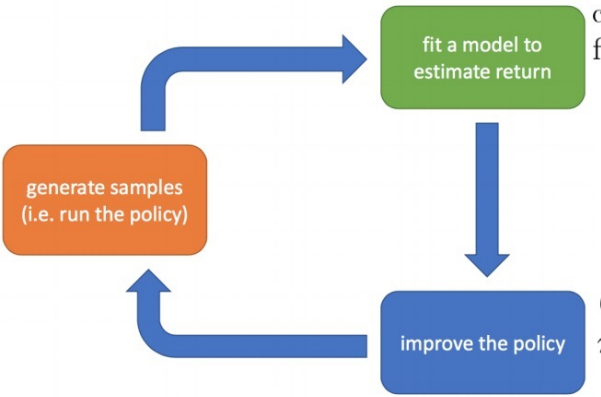


compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

fit $Q_\phi(\mathbf{s}, \mathbf{a})$ (actor-critic, Q-learning)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)

Reinforcement Learning Problem



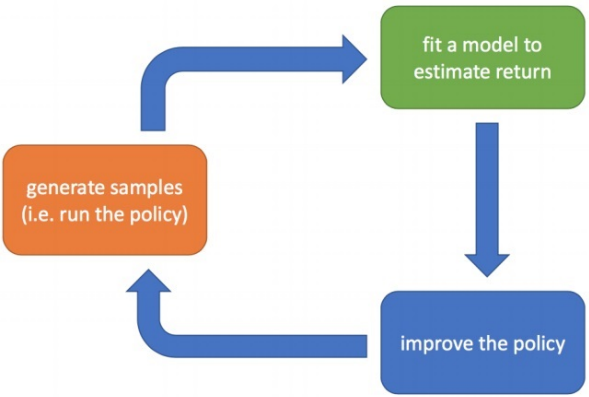
compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

fit $Q_\phi(\mathbf{s}, \mathbf{a})$ (actor-critic, Q-learning)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)

$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$ (Q-learning)

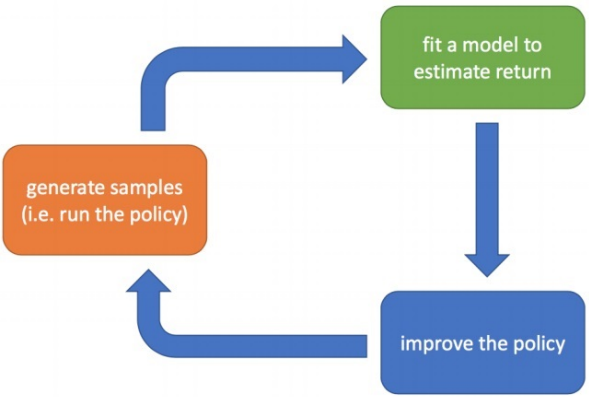
Reinforcement Learning Problem



compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)
fit $Q_\phi(\mathbf{s}, \mathbf{a})$ (actor-critic, Q-learning)
estimate $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)
$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$ (Q-learning)

Reinforcement Learning Problem



compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)
fit $Q_\phi(\mathbf{s}, \mathbf{a})$ (actor-critic, Q-learning)
estimate $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)
$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$ (Q-learning)
optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

Model-based vs Model-free

- In Model-free RL, we ignore the model. We perform sampling and simulation to estimate rewards
- In Model-based RL, if we can define a cost function ourselves, we can calculate the optimal actions using the model directly

Model Based

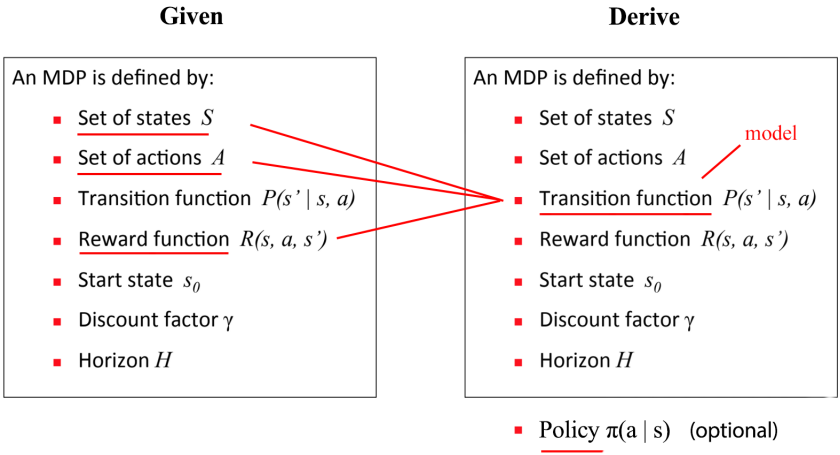

Given
Derive
An MDP is defined by:
Set of states S
Set of actions A
Transition function P(s' | s, a)
Reward function R(s, a, s')
Start state s_0
Discount factor γ
Horizon H
An MDP is defined by:
Set of states S
Set of actions A
model
Transition function P(s' | s, a)
Reward function R(s, a, s')
Start state s_0
Discount factor γ
Horizon H
Policy π(a | s) (optional)

Table of Contents

Model-based Reinforcement Learning

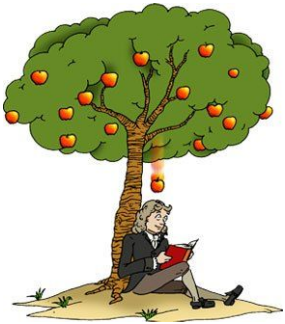
- In many games, like GO, chess – the rule of the game is the model

Model-based Reinforcement Learning

- In many games, like GO, chess – the rule of the game is the model
- In other cases, it can be the law of Physics

Model-based Reinforcement Learning

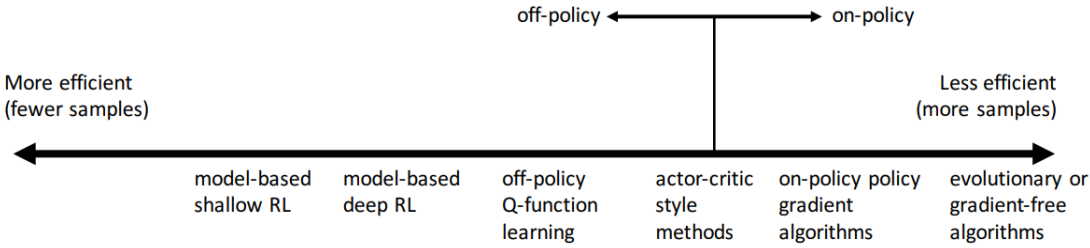
- In many games, like GO, chess – the rule of the game is the model
- In other cases, it can be the law of Physics



Why Model-Based?

- Sample efficiency

Sample Efficiency



Why Model-Based?

- Sample efficiency
- Transferability & generality. A model can be reused for achieving different tasks.

Model-based RL Formulation

- Consider a MDP with a set of states S, a set of actions A, and a state transition distribution $p(s'|s, a)$
- The goal of RL is to select actions in such a way as to maximize the expected sum of rewards over a trajectory

Model-based RL Formulation

- Consider a MDP with a set of states S, a set of actions A, and a state transition distribution $p(s'|s,a)$
- The goal of RL is to select actions in such a way as to maximize the expected sum of rewards over a trajectory
- **Model-based RL:**
  - Learning an approximate model $\hat{p}_\theta(s'|s,a)$ or $\hat{f}_\theta(s,a)$, parameterized by θ, that approximates the unknown transition distribution $p(s'|s,a)$ of the underlying system dynamics. The parameters θ can be learned to maximize the log-likelihood of observed data D.

Model-based RL Formulation

- Consider a MDP with a set of states S, a set of actions A, and a state transition distribution $p(s'|s,a)$
- The goal of RL is to select actions in such a way as to maximize the expected sum of rewards over a trajectory
- **Model-based RL:**
  - Learning an approximate model $\hat{p}_\theta(s'|s,a)$ or $\hat{f}_\theta(s,a)$, parameterized by θ, that approximates the unknown transition distribution $p(s'|s,a)$ of the underlying system dynamics. The parameters θ can be learned to maximize the log-likelihood of observed data D.
  - Learned model's predictions can then be used to
    - learn a policy, or

Model-based RL Formulation

- Consider a MDP with a set of states S, a set of actions A, and a state transition distribution $p(s'|s,a)$
- The goal of RL is to select actions in such a way as to maximize the expected sum of rewards over a trajectory
- **Model-based RL:**
  - Learning an approximate model $\hat{p}_\theta(s'|s,a)$ or $\hat{f}_\theta(s,a)$, parameterized by θ, that approximates the unknown transition distribution $p(s'|s,a)$ of the underlying system dynamics. The parameters θ can be learned to maximize the log-likelihood of observed data D.
  - Learned model's predictions can then be used to
    - learn a policy, or
    - perform online planning to select optimal actions

Model Based

Mathematically, the model predicts the next state.

$$x_t = f_\theta(x_{t-1}, a_{t-1})$$

- We can define this model with rules or equations

Model Based

Mathematically, the model predicts the next state.

$$x_t = f_\theta(x_{t-1}, a_{t-1})$$

- We can define this model with rules or equations
- We can model it, like using the Gaussian Process, Gaussian Mixture Model (GMM) or deep networks

Model Based

Mathematically, the model predicts the next state.

$$x_t = f_\theta(x_{t-1}, a_{t-1})$$

- We can define this model with rules or equations
- We can model it, like using the Gaussian Process, Gaussian Mixture Model (GMM) or deep networks
- To fit these models, we run a controller to collect sample trajectories and train the models with supervised learning.

Model Learning

- **Model-based control:** given an initial state s_0 estimate action sequence to reach a desired goal or maximize reward by unrolling the model forward in time
- **Model-based RL**

Model-based Control

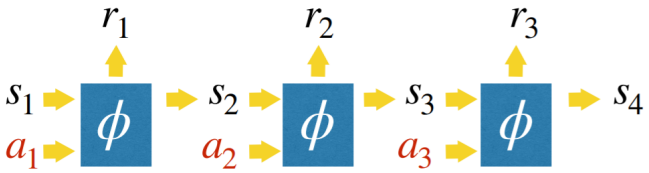
For model-based control we want either

1

$$\min_{a_1,\cdots,a_T} ||s_T - s*||, \text{ s.t. } \forall t, s_{t+1} = \hat{f}_\theta(s,a)$$

2

$$\max_{a_1,\cdots,a_T} \sum_{t=1}^{T} r_t, \text{ s.t. } \forall t, (s_{t+1}, r_{t+1}) = \hat{f}_\theta(s,a)$$



Model-based Control - SGD

1. Given an initial action sequence

$$a_1 \qquad a_2 \qquad a_3$$

Model-based Control - SGD

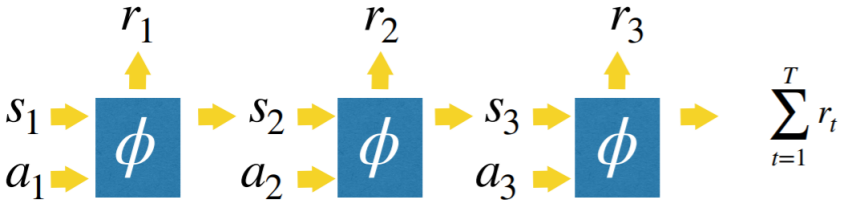
1. Given an initial action sequence
2. Unroll the model forward in time

Model-based Control - SGD

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and compute error against a desired final state

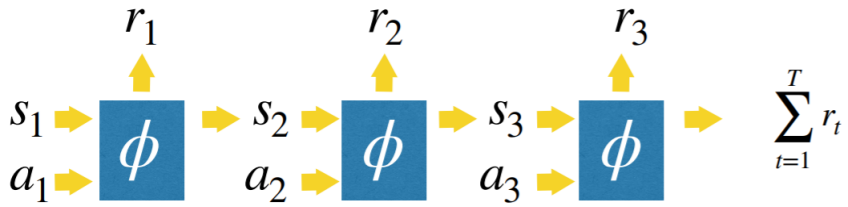
Model-based Control - SGD

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and compute error against a desired final state or compute sum of rewards



Model-based Control - SGD

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and compute error against a desired final state or compute sum of rewards
4. Backpropagate the error to the action sequence



Model-based Control - SGD

Does it work?

Model-based Control - SGD

Does it work?

Yes

PROBLEMS:

Model-based Control - SGD

Does it work?

Yes

PROBLEMS:

- Tiny errors accumulate fast along the trajectory

Model-based Control - SGD

Does it work?

Yes

PROBLEMS:

- Tiny errors accumulate fast along the trajectory
- The search space cab be too big for any base policy to cover full

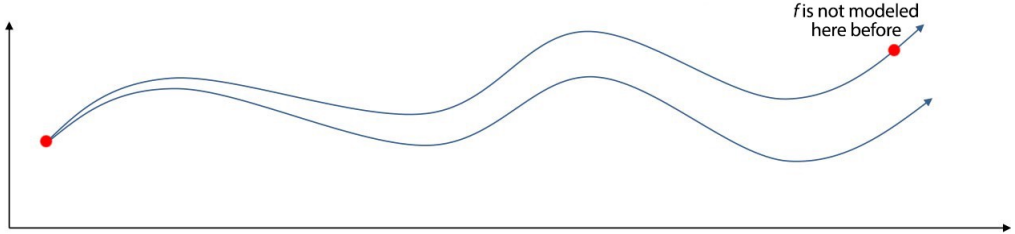
Model-based Control - SGD

Does it work?

Yes

PROBLEMS:

- Tiny errors accumulate fast along the trajectory
- The search space cab be too big for any base policy to cover full
- We may land in areas where the model has not been learned yet



Model Predictive Control

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and compute error against a desired final state or compute sum of rewards
4. Backpropagate the error to the action sequence

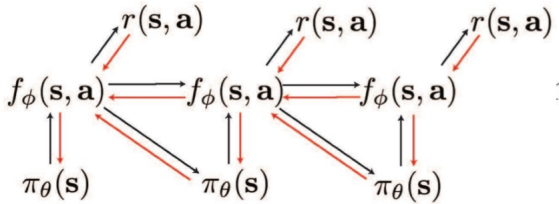
Execute only the first action and then GOTO 2, to avoid error accumulation. **(Model Predictive Control)**

Alternating between model and policy learning

Model-based RL: Backprop through model to optimize policy

Algorithm v0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i ||f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ into policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$



backpropagate

$$\max_\theta \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$$

Model-based RL: Backprop through model to optimize policy

Does it work?

Model-based RL: Backprop through model to optimize policy

Does it work? Yes

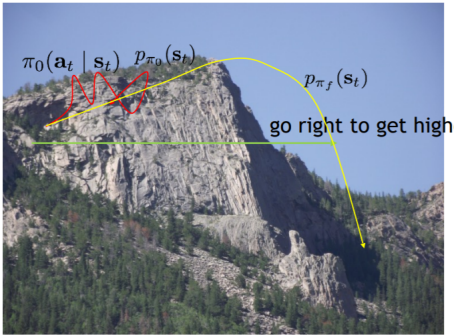
Model-based RL: Backprop through model to optimize policy

Does it work? Yes

- Essentially how system identification works in classical robotics
- Some care should be taken to design a good base policy
- Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters

Model-based RL: Backprop through model to optimize policy

Problem



1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum ||f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ into policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

$$p_{\pi_f}(\mathbf{s}_t) \neq p_{\pi_0}(\mathbf{s}_t)$$

Distribution mismatch problem becomes exacerbated as we use more expressive model classes

Model-based RL: Can we do better?

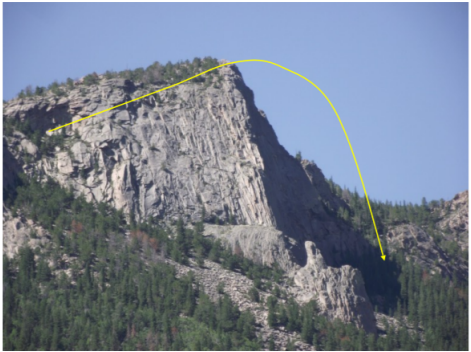
can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?

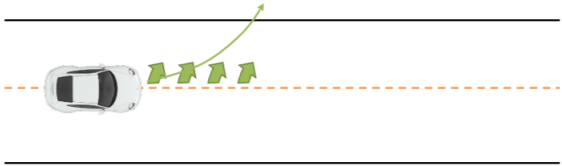
need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

Algorithm v1:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i ||f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ into policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

Model-based RL: What if we make a mistake?





Model-based RL: Model-predictive Control

Algorithm v2a:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i ||f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ to choose actions.
4. execute the first planned action, observe resulting state $\mathbf{s}'$
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

every N steps

model-predictive control (MPC)

Model-based RL: An alternative way to choose actions

every N steps

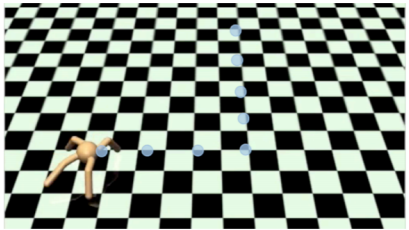
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i ||f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ to choose actions.
4. execute the first planned action, observe resulting state $\mathbf{s}'$
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

Can instead sample to choose actions:

A. Sample action sequences from some distribution (e.g. uniformly at random)

B. Run actions through model to prediction future

C. Choose action leading to the best future



Nagabandi et al. arXiv '17

Summary So Far

- Version 0: collect random samples, train dynamics, plan
  - Pro: simple, no iterative procedure
  - Con: distribution mismatch problem

Summary So Far

- Version 0: collect random samples, train dynamics, plan
  - Pro: simple, no iterative procedure
  - Con: distribution mismatch problem
- Version 1: iteratively collect data, refit model
  - Pro: simple, solves distribution mismatch
  - Con: might make mistakes with imperfect model

Summary So Far

- Version 0: collect random samples, train dynamics, plan
  - Pro: simple, no iterative procedure
  - Con: distribution mismatch problem
- Version 1: iteratively collect data, refit model
  - Pro: simple, solves distribution mismatch
  - Con: might make mistakes with imperfect model
- Version 2: iteratively collect data using MPC (replan at each step)
  - Pro: robust to small model errors
  - Con: computationally expensive, but have a planning algorithm available
- Two ways to optimize policy w.r.t. model:
  - backprop through model into policy
  - sampling-based optimization

Example: Google Brain SimPLe

Model Based Reinforcement Learning for Atari

Łukasz Kaiser [*1] Mohammad Babaeizadeh [*2 3] Piotr Miłoś [*4 5] Błażej Osiński [*4 5 3]
Roy H Campbell [2] Konrad Czechowski [4] Dumitru Erhan [1] Chelsea Finn [1] Piotr Kozakowski [4] Sergey Levine [1]
Afroz Mohiuddin [1] Ryan Sepassi [1] George Tucker [1] Henryk Michalewski [4 5]

Abstract

Model-free reinforcement learning (RL) can be used to learn effective policies for complex tasks, such as Atari games, even from image observations. However, this typically requires very large amounts of interaction – substantially more, in fact, than a human would need to learn the same games. How can people learn so quickly? Part of the answer may be that people can learn how the game works and predict which actions will lead to desirable outcomes. In this paper, we explore how video prediction models can similarly enable agents to solve Atari games with fewer interactions than model-free methods. We describe

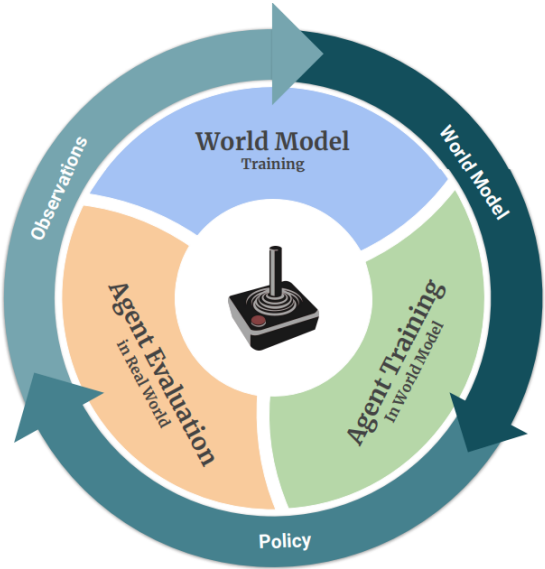
games so much faster? Perhaps part of the puzzle is that humans possess an intuitive understanding of the physical processes that are represented in the game: we know that planes can fly, balls can roll, and bullets can destroy aliens. We can therefore predict the outcomes of our actions. In this paper, we explore how learned video models can enable learning in the Atari Learning Environment (ALE) benchmark (Bellemare et al., 2015; Machado et al., 2017) with a budget restricted to 100K time steps – roughly to two hours of a play time.

Although prior works have proposed training predictive models for next-frame, future-frame, as well as combined future-frame and reward predictions in Atari games (Oh et al., 2015; Chiappa et al., 2017; Leibfried et al., 2016),

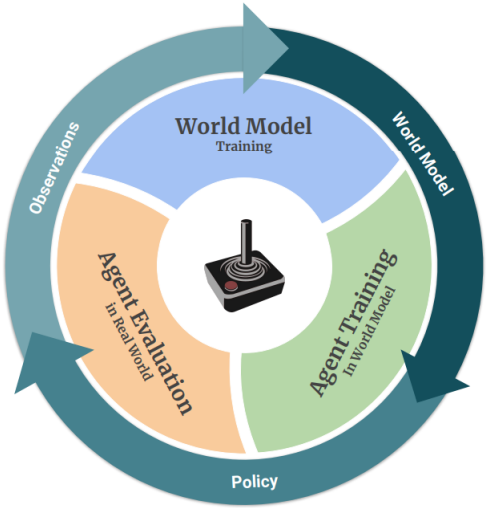
Example: Google Brain SimPLe

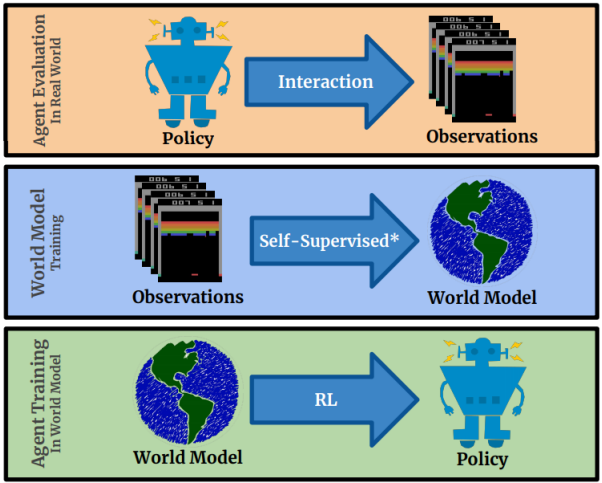
- SimPLe (Simulated Policy Learning) an entirely model-based deep reinforcement learning algorithm based on video prediction models
- Achieves competitive results on a series of Atari games
- Outperforms model-free algorithms in terms of learning speed on nearly all of the games
- The best model-free reinforcement learning algorithms require tens or hundreds of millions of time steps — equivalent to several weeks. SimPLe has obtained competitive results with only 100K interactions

Google Brain SimPLe: Main Loop



Google Brain SimPLe: Main Loop





Google Brain SimPLe: PseudoCode

Algorithm 1: Pseudocode for SimPLe

Initialize policy π
Initialize model environment env'
Initialize empty set D
while not done **do**
  ▷ collect observations from real env.
  $D \leftarrow D \cup \text{COLLECT}(env, \pi)$
  ▷ update model using collected data.
  $env' \leftarrow \text{TRAIN\_SUPERVISED}(env', D)$
  ▷ update policy using world model.
  $\pi \leftarrow \text{TRAIN\_RL}(\pi, env')$
end while

Google Brain SimPLe: PseudoCode

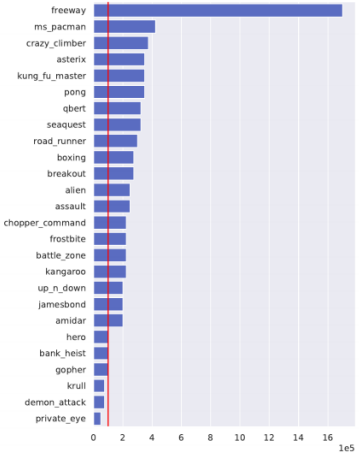


Figure 3. Comparison with Rainbow. Each bar illustrates the number of interactions with environment required by Rainbow to achieve the same score as our method (SimPLe). The red line indicates the 100K interactions threshold used by our method.

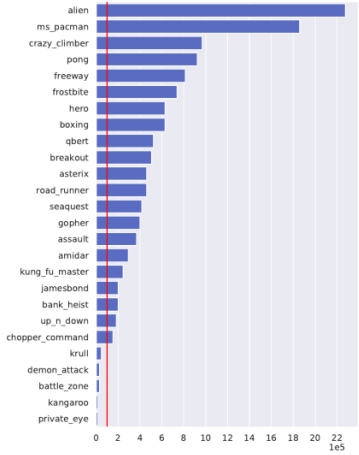
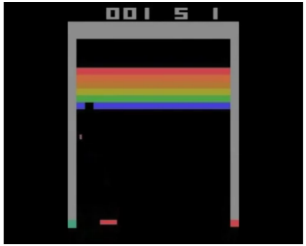
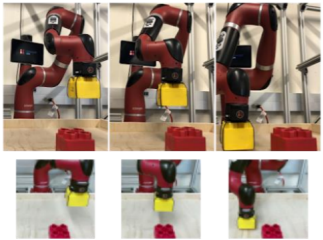


Figure 4. Comparison with PPO. Each bar illustrates the number of interactions with environment required by PPO to achieve the same score as our method (SimPLe). The red line indicates the 100K interactions threshold which is used by our method.

Complex Observation

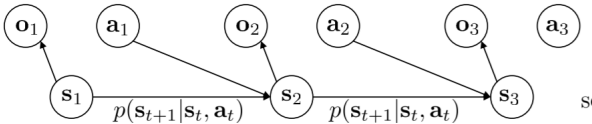




$$f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$$

What is hard about this?

- High dimensionality
- Redundancy
- Partial observability

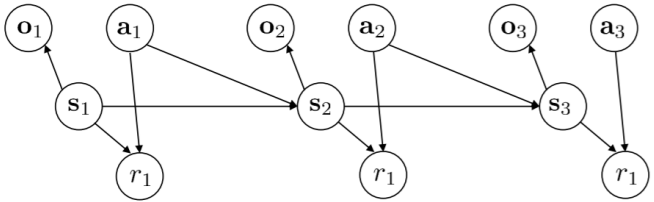


high-dimensional but not dynamic

low-dimension but dynamic

separately learn $p(\mathbf{o}_t|\mathbf{s}_t)$ and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$?

Learning in Latent Space



$p(\mathbf{o}_t|\mathbf{s}_t)$ observation model

$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ dynamics model

$p(r_t|\mathbf{s}_t, \mathbf{a}_t)$ reward model

How to train?

standard (fully observed) model: $\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$

latent space model: $\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} E\left[\log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i}|\mathbf{s}_{t,i})\right]$

expectation w.r.t. $(\mathbf{s}_t, \mathbf{s}_{t+1}) \sim p(\mathbf{s}_t, \mathbf{s}_{t+1}|\mathbf{o}_{1:T}, \mathbf{a}_{1:T})$

Learning in Latent Space

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{o}_t)$ (e.g., exploratory policy) to collect $\mathcal{D} = \{(\mathbf{o}, \mathbf{a}, \mathbf{o}')_i\}$
2. learn latent embedding of observation $\mathbf{s}_t = g(\mathbf{o}_t)$ and dynamics model $\mathbf{s}' = f_\phi(\mathbf{s}, \mathbf{a})$
3. use model $f_\phi(\mathbf{s}, \mathbf{a})$ to optimize policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|g(\mathbf{o}_t))$, appending visited tuples $(\mathbf{o}, \mathbf{a}, \mathbf{o}')$ to $\mathcal{D}$

What is reward for optimizing policy?

reward signal: $r(\mathbf{o}, \mathbf{a}) = r(\mathbf{a}) + ||g(\mathbf{o}) - g(\mathbf{o}_{\text{goal}})||$

Challenges in Model Learning

- Under-modelling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth

Challenges in Model Learning

- Under-modelling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth
- Over-fitting: If the model class is very expressive (e.g., neural networks) the model will overfit, especially in the beginning of training, where we have few samples

Challenges in Model Learning

- Under-modelling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth
- Over-fitting: If the model class is very expressive (e.g., neural networks) the model will overfit, especially in the beginning of training, where we have few samples
- Errors compound through unrolling

Challenges in Model Learning

- Under-modelling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth
- Over-fitting: If the model class is very expressive (e.g., neural networks) the model will overfit, especially in the beginning of training, where we have few samples
- Errors compound through unrolling
- Need to capture different futures (stochasticity of the environment)

Challenges in Model Learning

- Under-modelling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth
- Over-fitting: If the model class is very expressive (e.g., neural networks) the model will overfit, especially in the beginning of training, where we have few samples
- Errors compound through unrolling
- Need to capture different futures (stochasticity of the environment)
- Need to represent uncertainty outside of the training data

Model-Based vs. Model-Free Algorithms

MODEL-BASED

- Pros:
  - Easy to collect data in a scalable way (self-supervised)
  - Possibility to transfer across tasks
  - Typically require a smaller quantity of supervised data
- Cons:
  - Models don't optimize for task performance
  - Sometimes harder to learn than a policy
  - Often need assumptions to learn complex skills (continuity,resets)

MODEL-FREE

- Pros:
  - Makes little assumptions beyond a reward function
  - Effective for learning complex policies
- Cons:
  - Require a lot of experience (slower)
  - Not transferable across tasks

Notes

Model-based RL is an under-explored area of research
Two active, exciting areas:

- model-based approaches with high-dimensional observations
- combining elements of model-based planning & model-free policies

Example: Learning Dexterous Manipulation

Deep Dynamics Models for Learning Dexterous Manipulation

Anusha Nagabandi, Kurt Konoglie, Sergey Levine, Vikash Kumar
Google Brain

Abstract: Dexterous multi-fingered hands can provide robots with the ability to flexibly perform a wide range of manipulation skills. However, many of the more complex behaviors are also notoriously difficult to control: Performing in-hand object manipulation, executing finger gaits to move objects, and exhibiting precise fine motor skills such as writing, all require finely balancing contact forces, breaking and reestablishing contacts repeatedly, and maintaining control of unactuated objects. Learning-based techniques provide the appealing possibility of acquiring these skills directly from data, but current learning approaches either require large amounts of data and produce task-specific policies, or they have not yet been shown to scale up to more complex and realistic tasks requiring fine motor skills. In this work, we demonstrate that our method of online planning with deep dynamics models (PDDM) addresses both of these limitations; we show that improvements in learned dynamics models, together with improvements in online model-predictive control, can indeed enable efficient and effective learning of flexible contact-rich dexterous manipulation skills – and that too, on a 24-DoF anthropomorphic hand in the real world, using just 4 hours of purely real-world data to learn to simultaneously coordinate multiple free-floating objects. Videos can be found at https://sites.google.com/view/pddm/

Keywords: Manipulation, Model-based learning, Robots